SCHOOL OF ENGINEERING - STI
ELECTRICAL ENGINEERING INSTITUTE
SIGNAL PROCESSING LABORATORY

Jean-Paul Wagner and Pascal Frossard

EPFL - FSTI - IEL - LTS
Station 11
Switzerland-1015 LAUSANNE

Phone: *+4121 6934709*
Fax: *+4121 6937600*
e-mail: `{jean-paul.wagner,pascal.frossard}@epfl.ch`

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

MEDIA-FRIENDLY DISTRIBUTED RATE ALLOCATION

Jean-Paul Wagner and Pascal Frossard

Ecole Polytechnique Fédérale de Lausanne (EPFL)
Signal Processing Laboratory

Technical Report LTS-2008-003

March 25, 2008

Part of this work has been submitted to IEEE Transactions on Multimedia.
This work has been partly supported by the Swiss Innovation Promotion Agency, under grant CTI - 7388.2 ESPP-ES.

Media-Friendly Distributed Rate Allocation

Jean-Paul Wagner and Pascal Frossard
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Signal Processing Laboratory - LTS4
Switzerland - 1015, Lausanne

***Abstract*—The paper addresses the problem of distributed congestion control in networks where several video streaming sessions share joint bottleneck channels. The proposed algorithm takes into account the specificities of the video streams, as well as the requirements of the heterogeneous streaming clients, in order to determine the actual benefit of additional channel resources, or equivalently the utility of rate increments. The utility-based congestion control framework initially proposed by Kelly [1], [2] is extended to cope with heterogeneous delays in the network as well as with the specific requirements of video streams. We describe an original implementation of such a distributed congestion control algorithm for scalable video streams, using the common RTP/UDP/IP protocol stack, where receiver feedback triggers the adaptation of the streaming rate at each server independently. Finally, we provide extensive simulation results that demonstrate the performance of the proposed solution, which successfully distributes the network resources among the different sessions in order to maximize the average video quality. The proposed scheme is also shown to cohabit fairly with TCP, which is certainly an important advantage in today's network architectures.**

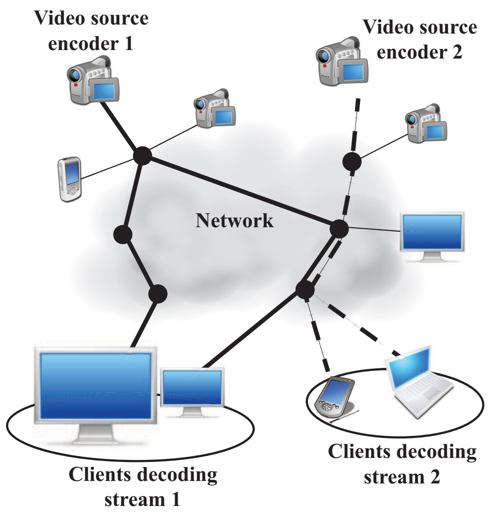

Fig. 1. System view. The goal is to deliver the best possible media quality to the set of clients, while keeping the network in a stable state in dynamic scenarios.

I. Introduction

The predominance of FTP or HTTP traffic and the scalability of the Internet rely largely on the success of the transport protocol that controls the vast majority of the Internet connections, the Transport Control Protocol (TCP) [3]. One of the design goals of the TCP algorithm is to fairly distribute the bandwidth resources among the different concurrent flows. TCP provides a very efficient and distributed congestion control solution for low-bandwidth data streams, or elastic flows that do not present strict delay constraints.

While these types of data have formed the bulk of the Internet traffic, the Internet is evolving into a transport medium for the distribution of data such as audio and video streams due to the emergence of attractive multimedia applications. Multimedia streaming applications impose different requirements than those underlying FTP or HTTP traffic. On the one hand, the media streams have to be delivered at a rather high and sustained rate in order to deliver an acceptable Quality of Service (QoS) to the media client. On the other hand, an encoded media stream generally carries significant redundancy and therefore presents some inherent resilience to packet loss. The experience of the the end user is mostly driven by the delay and the fluctuations of quality in media streaming applications. Unsurprisingly, TCP does not perform very well in regulating the rate of media streams since it has not been planned for controlling such data flows. In particular, the typical sawtooth behavior in the controlled transmission rate profiles and the delays experienced by some packets due to the retransmission and exponential back-off policies are not ideal for media streaming. In addition, the importance of the media packets is quite heterogeneous between the different media streams, and even within a given media stream. TCP typically cannot consider these properties and rather targets a fair bandwidth distribution, which does not guarantee optimized average performance in multimedia communications.

In this paper we will explore an alternative rate- and congestion control algorithm for a bandwidth resources allocation that is adapted to the characteristics of media streams. Instead of compensating the drawbacks of TCP through rather complex scheduling and retransmission schemes, we consider an orthogonal approach and propose a *media-friendly* control algorithm, which adapts the rate in the network to the needs of the media streams. Our work is based on a class of congestion control algorithms that have been first proposed by Kelly [1], [2]. Their general objective is to maximize the aggregate *utilities* the end users retrieve from their network usage. Given this objective, a candidate algorithm should drive the system close to its optimal state, in which the network utilization is high and where each client receives a rate that is useful for the media streaming application. The algorithm has also to be run in a distributed way in order to guarantee the scalability of the system. The family of congestion control algorithms described in [1], [2] have the benefit of i) potentially allocating smooth rate profiles without abrupt transitions along time and

ii) distinguishing each flow based on a *utility* criterion. This criterion can typically capture the benefit of an increment of bandwidth for a given media streaming client. In the framework of this paper, it depends on the proper characteristics of a media stream and on the particular requirements of the application.

We consider scenarios described by the framework illustrated in Figure 1. The media sources are captured and encoded in various points at the edges of an overlay network. Clients with heterogeneous needs in terms of frame rate or display resolution capabilities for example can connect to any media source from anywhere in the network. Such a scenario could typically represent a video-surveillance system with both high-end decoding stations and low power mobile clients. The objective of the distributed congestion control algorithm presented in this paper is to jointly achieve network stability and optimized average quality for all decoding clients. In addition, the algorithm has to dynamically adapt to changes in the topology in order to accommodate new clients, or rather to efficiently use increased bandwidth resources.

Our solution is based on the utility-based congestion control developed by Kelly [1], [2]. We extend these algorithms to practical systems with heterogeneous feedback delays, where we show that the algorithms remain stable. Then we compute utility curves for scalable video streams, which permit to define several levels of quality by changing the spatial, temporal or SNR resolutions. We design a client feedback mechanism in order to infer the network state. We finally implement the distributed congestion control scheme for various streaming scenarios where H.264/SVC streams are transmitted using RTP/UDP/IP protocols. We provide extensive simulation results that demonstrate the convergence of the distributed congestion control algorithm, even in dynamic environments. The results also show that the bandwidth is properly distributed among the clients such that the average quality is optimized. Finally, the proposed algorithm is shown to cohabit fairly with TCP flows.

The remainder of this paper is organized as follows. In Section II we provide some background on Kelly's theoretical framework. We extend it in Section III with an important stability proof. In Section IV we outline the proposed distributed congestion control algorithm and show how we can incorporate media-friendliness through a judicious choice of Utility functions. In Section V we describe the careful and fully scalable implementation of the proposed algorithm that can significantly boost the performance of the congestion control in practical scenarios. In Section VI we validate our findings through extensive NS-2 simulations. Finally we conclude with Section VII.

II. Background

In this section, we provide some background and references on Kelly's Network Utility Maximization (NUM) problem that has been first stated in [2]. We give first a brief overview on distributed algorithms that solve the aforementioned problem. Then we provide a stability proof that is of high importance for the practical framework studied in this paper. Namely, we show that a system controlled using the NUM framework is stable in the case of general rate utility functions and arbitrary round-trip delays. Finally, we discuss the *fairness* between flows resulting from the rate allocations computed in the NUM framework.

A. Network Utility Maximization

Let $s_i(t)$ be the rate assigned to flow i at time t. Further, assume that we assign to each flow i a *utility function*. This function describes by a utility value $U_i(s_i(t))$ the usefulness of the streaming rate $s_i(t)$ for an application that is served by flow i [4]. We suppose that the utility functions are continuous functions of the rate and that they are concave. Let $\mathcal{I}$ be the set of all flows in a given network, characterized in turn by a set of network links l with their respective capacity constraints.

The original NUM problem consists in finding for every time t a set of rates $s_i(t)$ that maximizes the sum of utilities for all flows, $\sum_{i\in\mathcal{I}} U_i(s_i(t))$, while satisfying the rate constraints on each link in the network. The direct solution of this optimization problem is difficult to find for large networks, and it further assumes that a central entity controls the rate of each flow. Instead, Kelly has proposed in [2] to solve a relaxed version of the original problem, by adjusting the rate of the flows in order to satisfy the following differential equations at any time t:

$$\frac{d}{dt}s_i(t) = \kappa_i s_i(t)\left(U_i'(s_i(t)) - p_i(t)\right). \qquad (1)$$

The term $s_i(t) \cdot U_i'(s_i(t))$ is referred to as the *willingness to pay* for the resource (i.e., the available bandwidth) and $p_i(t)$ is a pricing function, updated by the network, which indicates the price of the offered resource. In practice, $p_i(t)$ is often a congestion indication function that is fed back to the controller from either the network or the end user and κ_i is a constant gain factor.

Note that this system of differential equations drives each rate $s_i(t)$ into a steady state in which there is an equilibrium between the willingness to pay, and the flow's contribution to the resource's price. The former depends on the used utility function $U_i(s_i(t))$ through its gradient, while the latter depends on the current network state, which is expressed by the congestion indication function $p_i(t)$. It is worth noting that Equation (1) can be straightforwardly discretized and turned into a rate update equation that leads to a practical implementation of a distributed control algorithm for solving the original NUM problem. To date, this has yielded the most promising implementations of control algorithms, even if there are other ways of solving the original NUM problem in a distributed way [5].

B. Fairness

The control algorithms based on Kelly's framework provide a fair distribution of the network resources among the different competing flows in the network. The fairness characteristic might however be defined in several different ways. In particular, the fairness could be linked to the distribution of the network bandwidth, or rather to the distribution of network resources that balance utilities among flows. We refer to [6], [7] for a detailed study on the fairness in Kelly's framework. For the sake of completeness we provide a short summary here:

- If the controller relies on end-to-end feedback measures only, the control algorithm provides *proportional fairness*, meaning that flows congesting multiple routers are allocated less bandwidth in the stable state than flows congesting less routers.
- If the controller has access to the congestion levels of each router, which is however not a realistic scenario in today's Internet, the controller can adjust the rate for each flow according to the most congested router, thus providing *max-min fairness* [8].
- If additionally each flow is characterized by a potentially different utility function, the resulting rate allocations are weighted by the respective utility values in the stable state.

The practical system that we propose in this paper relies on end-to-end measures and uses different utility function for each flow. Hence, the control algorithm based on Kelly's framework provides in this case a *utility-proportional* fairness with the streaming rates allocated to the different flows.

C. Stability results

An important characteristic of any distributed control algorithm lies in the stability of the system. Both stability and convergence have been extensively studied for systems based on the differential equations given in (1) (see for example [2], [9], [10], [11] and references therein). However, the ideal system described above has only marginal practical relevance since it assumes that network or client feedbacks are immediately available at each source in order to update the rate of the corresponding flow. In practical systems, each flow i has a fixed starting point (the sender) as well as a fixed end point (the receiver) in the network. However, the route that connects the two may change in time, which therefore affects the round-trip delay between sender and receiver. Moreover, the congestion level of each router that is traversed by the flow is dynamically changing. The delay experienced by feedback information is not only heterogeneous across flows in the network, but it has also a random distribution for each flow. When the system experiences some delays in the feedback loops, the differential equations of relation (1) read as:

$$\frac{d}{dt}s_i(t) = \kappa_i \left(s_i(t)U_i'(s_i(t)) - s_i(t - D_i^R)p_i(t - D_i^B) \right), \tag{2}$$

In this case, D_i^R is the round-trip delay and D_i^B is the delay on the back-trip from the point in the network that created the feedback to the sender. Note that the congestion indication function $p_i(t - D_i^B)$ is synthesized D_i^B time units earlier, and that it relates to the rate allocated by the controller D_i^R time units earlier. In the last years, a substantial amount of research has been devoted to providing stability results for controllers with delayed feedbacks. For example the authors in [9] provide results for the case of equal round-trip delays for each flow. More recently, authors in [12] have studied the stability of systems with arbitrary but constant round-trip delays. Further stability results are provided in [13], [14], [15], [16], [17], [8] and references therein. In each of these works, the scenario corresponds to particular applications: either each flow uses the same utility function, or the delays are different for the various flows but constant in time, or the parameters of the rate update equation are dynamically adjusted with respect to the experienced delay. The latter implies that a precise measurement of the experienced round-trip delay is available at the controller, so that oscillations can be avoided in the system. The application considered in this paper however does not correspond to any of these scenarios, and we therefore extend the stability proof to a more generic case in the next section.

III. Stability with random feedback delays

The application we are targeting specifically calls for stability results for the scenario in which *general* concave utility functions (i.e. they are different for each media stream) are used, and in which the round-trip delays are arbitrary. In [18], the author provides an elegant stability proof for the case where general utility functions are used, and where the round-trip delays in the network are arbitrary across flows, but constant in time. The author conjectures that the system remains stable if the round-trip delays take on arbitrary values. In what follows we borrow the framework from [18] and we extend the stability result to the case where generic, concave utility functions are used and where the feedback is arbitrarily delayed for each flow.

We consider a network made up of L links, indexed by l. We call $\Delta_{l,i}$ the Round-Trip delay experienced by data transmitted from source i to traverse link l and get back to the source. Hence the experienced Round-Trip delay D_i^R for user i, takes on values in the set $\{\Delta_{1,i}, \ldots, \Delta_{L,i}\}$, depending on the receiver of flow i. Further we denote by Δ_l the maximum Round-Trip delay for data to get sent from any source, traverse link l and get back to the source: $\Delta_l = \sup_i\{\Delta_{l,i}\}$. Let $\overline{D}$ be an upper bound on all experienced Round-Trip delays in the network: $\overline{D} = \sup_i\{D_i^R\}$. Clearly, the following relation holds:

$$\overline{D} \geq \Delta_l, \quad 0 \leq l \leq L. \tag{3}$$

The following Lemma is proven in [18] and is reproduced here as it is of crucial importance in the reasoning that follows.

Lemma 1: Let A and B be two matrices, the entries of which, indexed by couples (n, m), all satisfy

$$|A_{n,m}| \leq B_{n,m} \tag{4}$$

and hence, the $B_{n,m}$ are real, nonnegative. Then, the spectral radius, i.e., the largest positive eigenvalue of A, is smaller or equal to that of B.

The main result of [18] states that the system of delayed differential equations (2) is asymptotically stable if $\kappa_i > 0$, $U_i(\cdot)$ is a concave function for each i, and if all the Round-Trip delays in the network coincide with a single scalar: $D_i^R = D, \forall i$. A sufficient condition for this to be the case is that the spectral radius of matrix N, given as

$$N = D \sum_{0 \leq l \leq L} M^{(l)} \tag{5}$$

is smaller than 1. This is shown to be the case *for any scalar D*, if the matrix M, which does not depend on the Round-Trip delays, is positive definite. This condition is in turn verified if $\kappa_i > 0$ and the utility functions $U_i(\cdot)$ are concave. We now extend this stability result to arbitrary feedback delays.

Theorem 1: Assume that the Round-Trip delays D_i^R take on arbitrary values bounded by the scalar $\overline{D}$. Then the system given in Eq. (2) is asymptotically stable under the assumptions that $\kappa_i > 0$ and the Utility functions $U_i(\cdot)$ are concave for all i.

Proof: Following the same reasoning as in [18], a sufficient condition for this to be true is that the spectral radius of matrix N' is smaller than 1, with

$$N' = \sum_{0 \leq l \leq L} \Delta_l M^{(l)}, \tag{6}$$

and M the same as in the previous development. Let us introduce the following matrix:

$$\overline{N} = \overline{D} \sum_{0 \leq l \leq L} M^{(l)}. \tag{7}$$

From Eq. (5) we know that the spectral radius of $\overline{N}$ is less than 1. Using Eq. (3) we can further bound the elements of matrix N' as follows:

$$N' \leq \sum_{0 \leq l \leq L} \overline{D} M^{(l)} \tag{8}$$

As the spectral radius of the right-hand side of (8) is smaller than 1, the proof of this theorem is concluded by the application of Lemma 1. ■

This result of stability is important for the design of the congestion control algorithm proposed in the next section. It is moreover most relevant to any practical system as in practice the Round-Trip delays that are observed are always bounded.

IV. Distributed Rate Allocation Algorithm

We propose now a distributed rate allocation algorithm for video sequences, based on the utility maximization framework described above. We first present a discretized version of the system of differential equations given in the relation (2), which leads to discrete-time rate update equation. Then we propose utility functions that are adapted to practical scenarios for video streaming.

A. Rate update equation

The distributed control algorithm can only act at discrete time instants in practice. An approach based on finite differences can be used to obtain a discrete-time version of Eq. (2), as suggested in [9]. This results in the following update equation:

$$\begin{aligned} s_i(t) = s_i(t-1) + \kappa_i \big(& s_i(t-1) U_i'(s_i(t-1)) \\ & - s_i(t - D_i^R) p_i(t - D_i^B) \big), \end{aligned} \tag{9}$$

The congestion indication function (pricing function) for flow i can be computed based on the *received* rate $r_i(t - D_i^B)$ measured at the client at time $t - D_i^B$. In this case, it reads

$$p_i(t - D_i^B) = \frac{s_i(t - D_i^R) - r_i(t - D_i^B)}{s_i(t - D^R)}. \tag{10}$$

Note that there is a temporal shift between the time at which the reference sending rate, the actual sending rate, and the received rate are computed due to delays in the system. This may delay the convergence of the system. Using Eq. (9), the sender updates the rate $s_i(t)$ using the last taken control $s_i(t-1)$ as reference rate and and a delayed feedback term. This feedback represents pricing information about the control taken D_i^R time units earlier. For example, if there is no congestion, the received rate is equal to the sending rate at time $t - D_i^R$. In that case the update equation will lead to a pure increase of the rate. In the event of a congestion, only part of the sending rate is received so $r_i(t - D_i^B) < s_i(t - D_i^R)$. Hence the price of the resource for flow i is increased and the rate will be re-adjusted downwards accordingly.

In order to cope with delays and improve the stability of the system, we can rather use $s_i(t - D_i^R)$ as the reference rate for the update equation [14], [19], so that the temporal drift between reference rate and pricing function is virtually cancelled. However, differently from [19] we do not intend to use a constant *willingness to pay* throughout the network, as the rate allocation should reflect the relative utilities of the various streams. Hence, in order to eliminate the temporal drift between the term of Eq. (9) that increases the reference rate and the feedback term that penalizes the reference rate, we need to evaluate the Utility function at the rate given by $s_i(t - D_i^R)$ as well. Finally, the proposed controller use the following discrete-time rate update equations:

$$\begin{aligned} s_i(t) = s_i(t - D_i^R) + \kappa_i s_i(t - D_i^R) & \\ \left[U' \left(s_i(t - D_i^R) \right) - p_i(t - D_i^B) \right]. & \end{aligned} \tag{11}$$

B. Role of Utility Functions

In the analysis about the stability of the distributed rate allocation algorithm, the only assumption on the continuous utility functions relies in their concavity. It leaves quite some flexibility in the choice of these functions, such that they correspond to the characteristic of the target application. In particular, the choice of distinct utility functions for each stream directly influences the rate allocation among all streams congesting a bottleneck in the stable state. From a networking point of view, it can be inferred from the rate update equation (11) that the average experienced loss rate $p_i(\cdot)$ is equal to the gradient value of the utility function, evaluated at the rate operation point that is reached in the stable state. The controller then maximizes the system's aggregate utility by iteratively allocating more rate to streams that have a larger benefit at the current rate operation point.

As the normalized congestion signal $p_i(\cdot)$ takes on values in the interval $[0, 1]$, the utility gradient values of any stream in the system are normalized by a system-wide constant. This constant corresponds to the largest gradient value any used utility function throughout the system can take on. It is important to note that the normalization constant should not be stream-dependant in order to avoid distortions between the relative utility curves assigned to different streams. The maximum gradient value in the system also drives the maximum observed stable-state loss rate per stream throughout the system. Hence, by correctly scaling all the gradients, we can bound the maximum loss rate π to the maximum value that can be tolerated by the target application. Hence, we can finally rewrite the Eq. (11) as:

$$s_i(t) = s_i(t - D_i^R) + \kappa_i s_i(t - D_i^R) \left[\pi \cdot U'\left(s_i(t - D_i^R)\right) - p_i(t - D_i^B)\right]. \quad (12)$$

C. Video Utility

We describe now in more detail the choice of utility functions for video streaming applications. An obvious choice is to relate the utility of a media stream to the rate-distortion characteristics of the encoded sequence. In that case the above control algorithm iteratively allocate the rates among streams proportionally to their contribution to the average video quality computed on all the streams.

We focus in the rest of this paper on video encoders that provide the possibility to generate traffic that can be tuned to achieve any rate within a bounded rate interval. Such encoders include for example Motion-JPEG2000, in which each frame is progressively intra-coded: the rate of each frame can thus be adapted by selectively dropping wavelet coefficients, while increasing the distortion gracefully. Another example is given by the progressive refinement (PR) slices of the scalable video coding (SVC) amendment to the H.264/MPEG-4 AVC standard. Using SVC, a video can be decoded at the full encoding rate, yielding the highest possible decoded quality in terms of SNR, framerate and resolution. Aside from this, one has the option to decode a number of sub-streams that have been specifically included while encoding. Each of the sub-streams represents a version of the video that is degraded in either SNR, frame rate or spatial resolution, or any combination of these. Typically it comes down to the targeted application or to the capabilities of the decoding client to decide which sub-stream is most useful. Finally, each of these sub-streams can be encoded using progressive refinement slices, providing fine granularity scalability (FGS). These slices can be cut at any point in order to finely tune the rate within a substream, while gracefully increasing the distortion. Both of the aforementioned encoding choices rely on fine grained rate adaptation that results in SNR scalability. Hence they are able to generate the kind of traffic we aim for: any rate within a bounded rate interval can be achieved. As this results in SNR scalability, the resulting rate-distortion curves over that interval are concave and can be used as utility functions for the respective streams.

Examples of the utility functions used in this paper are illustrated in Figure 2 for a number of test sequences at different framerates and resolutions. They have been computed on video sequences encoded using the H.264 SVC extension and using PR slices. The sequences have first been segmented into Groups of Pictures (GOPs) of equal size, and each GOP has been encoded independently into PR slices. The utility functions for the complete sequence have finally been extracted by decoding each GOP at a number of fixed rate points, and by averaging the resutling Y-PSNR value at each rate point for all the GOPs of the sequence. Note that we do not rely on an analytical model to specify the utility curves for each stream contrarily to [20], [21], but we rather use the exact rate-distortion information.

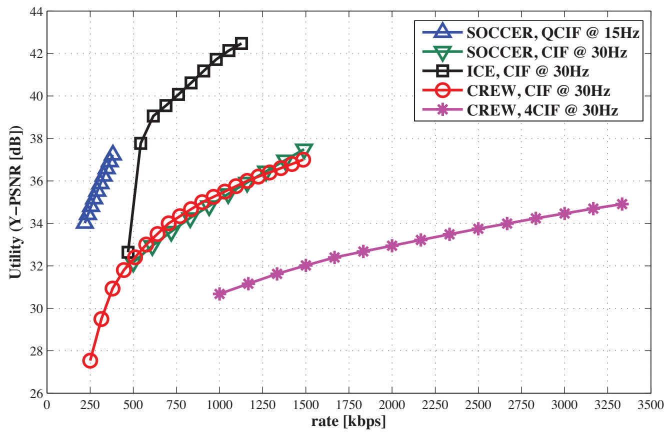

Fig. 2. Utility curves can reflect the rate-distortion characteristics. Here we show the rate-distortion curves of several H.264-SVC(FGS) encoded test sequences (SOCCER, CREW, ICE).

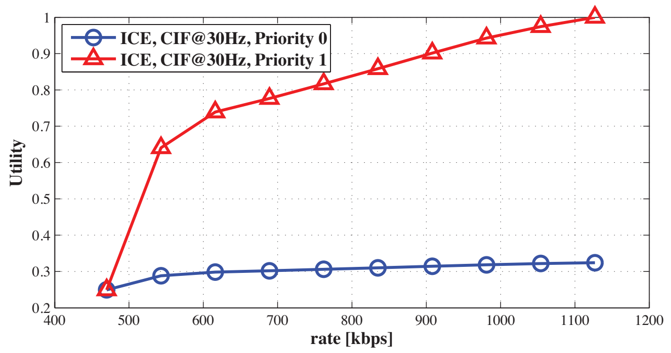

Fig. 3. Utility curves can also reflect traffic prioritization. Here Priority Class 0 uses the (scaled) R/D information for the SOCCER sequence, whereas users in Priority Class 1 use a utility function that has a gradient five times as large at each rate point.

Finally, it should be noted that our framework is very generic and that any function that is both concave and continuous is valid as a utility function. In particular, priority or service classes can also be incorporated in the utility functions. For example, we can design utility functions for 2 classes of streaming clients decoding the same sequence The rate for receivers in the lower Priority Class is adapted using to the rate-distortion function. The receivers in the higher Priority Class use a different utility function, whose gradient corresponds to a scaled version of the rate-distortion curve at any rate point. Clients in the higher Priority Class hence see the quality of their video streams be adapted faster and reach a higher level. Such utility functions are illustrated in Figure

3, where the utility gradient is multiplied by five for the high priority class.

V. Implementation

We propose now a practical implementation of the control system described above. We have shown that the distributed control algorithm is stable, even with heterogenous feedback delays, which are likely to happen in real scenarios. We outline scalable implementation of the proposed framework, and we explain in detail the design choices for the rate update equation and the distributed control in a video streaming system.

A. Design issues

From the above development, it is clear that the rate update equation (12) has ideally to be applied as often as possible in order to emulate a continuous time control system. This requires the availability of accurate feedback at each moment in time and at each end-point in the network. At the same time, it implies that the sending rate can be adapted at any time instant. However, when we are dealing with real video streams, it is clear that we cannot adapt the video rate at any arbitrary time instant. For example, dropping random parts of the bitstream results in large and uncontrolled losses of quality due to the inherent decoding dependencies between video elements. Scalable video coding provides an interesting solution for flexible adaptation of the bitstream. For example, encoding formats such as H.264-SVC(FGS) form independently decodable compressed units such as Groups of Pictures (GOP), which are typically groups of 16 or 32 frames. They further offer the possibility to extract a substream of a given rate from any independently decodable entity of the stream. The rate control has therefore to be performed on GOPs, and the rate update equation (12) of the control system should be synchronized with GOP boundaries. It is applied after each transmitted GOP, and defines the rate of the substream to be extracted from the next GOP.

The decision of the control algorithm relies on the feedback received from the clients, about the state of the streaming session. In addition, it uses the result of the decisions taken in previous iteration of the algorithm, as seen in the update equation (12). The controller has to know the sending rate that was computed $(t - D_i^R)$ time units earlier, where D_i^R is the *arbitrary* experienced Round-Trip delay. Maintaining the history of earlier decisions for each flow at each sender does however not provide a viable solution as it does not scale. In addition, it relies on very accurate Round-Trip Time measurements in order to avoid any drift between the sending rate that is actually used in the update equation and the feedback that is received. The utility function is evaluated on the sending rate, yielding the willingness to pay for the bandwidth that is offered, while the congestion signal that is fed back gives the price of the bandwidth resources. Any temporal drift between the computation of these values slows down the convergence of the distributed algorithm. A scalable system has therefore to avoid any temporal drifts, by gathering together the congestion signal, and the sending rate it corresponds to. In the next section, we propose a light-weight application layer protocol that respects the above constraints using the time-stamp information included in the transported video streams.

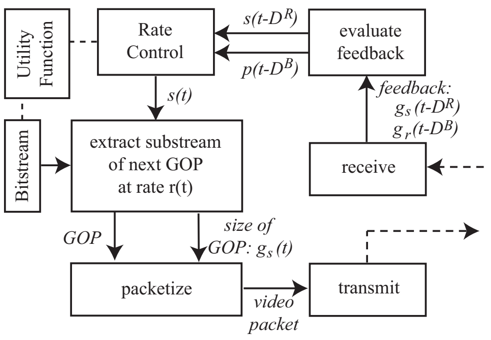

Fig. 4. Schematic view of our sender implementation.

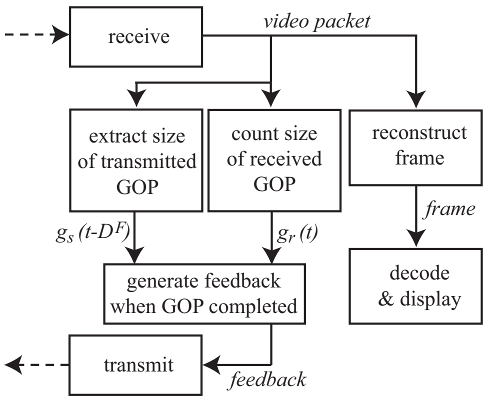
receive
video packet
extract size of transmitted GOP
count size of received GOP
reconstruct frame
g_s(t-D^F)
g_r(t)
frame
generate feedback when GOP completed
decode & display
transmit
feedback

Fig. 5. Schematic view of our receiver implementation.

B. Proposed control system

We propose now an implementation of the distributed control system at the application layer, for streaming scalable video over the classical RTP/UDP/IP protocol stack. For the sake of clarity we will drop the index i that specifies a particular flow and sender/receiver pair in what follows. We describe now in details the behavior of a client and sender pair, and all the concurrent streaming sessions in the system adopt the same strategy.

A schematic view of the sender implementation is first given in Figure 4. Upon reception of a feedback message, the rate controller given by Equation (12) updates the targeted sending rate to $s(t)$, which depends on the media stream that is being transmitted through the chosen utility function. The next GOP of the scalable bitstream is then optimally truncated so that its rate matches the target rate. The resulting substream is then packetized according to the video frames and Network Abstraction Layer (NAL) units and injected into the network. The exact size of the extracted GOP, denoted by $g_s(t)$, is added to the header of all the packets in the GOP. Note that that the sender knows the framerate f [Hz] of the stream as well as the length k of the GOP in frames, since those are usually negociated by the sender and the receiver. The sending rate is

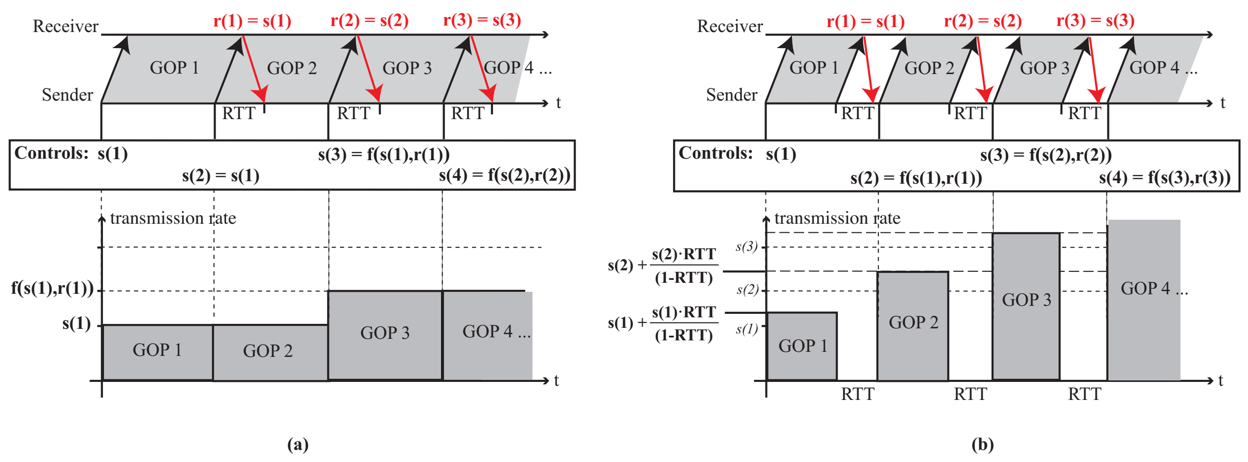

Fig. 6. Guard interval illustration. *Left:* When no guard interval is used, the available feedback is only considered in the control loop at a later stage. For example, the rate for GOP 3 is dependent on the rate and feedback of GOP 1. *Right:* By using a guard interval, the feedback can be integrated more rapidly in the control decisions.

therefore simply given by

$$s(t) = \frac{g_s(t)}{k} f \ . \tag{13}$$

The client behavior is then represented in Figure 5. It receives the packet stream and detects potential packet losses. It further reassembles the media bitstream and sends the reconstructed decodable parts of the bitstream to the decoder. At the same time, the receiver counts the number of bits $g_r(t)$ that are received for the current GOP. After the current GOP has been completely received, the receiver sends a feedback to the sender, where it includes the received size $g_r(t)$ as well as the GOP size $g_s(t - D^F) \geq g_r(t)$ that is read from packet headers. When the feedback eventually reaches the sender, it accurately reconstructs the control decision taken $D^R = D^F + D^B$ time units earlier. Even if the Round-Trip time D^R is arbitrary, the sender can replicate the past control decision and compute the previous sending rate as

$$s(t - D^R) = \frac{g_s(t - D^R)}{k} f. \tag{14}$$

Similarly it reconstructs the rate that is actually received by the client as:

$$r(t - D^B) = \frac{g_r(t - D^B)}{k} f \ . \tag{15}$$

The sender has thus access to all the information necessary to evaluate the congestion signal as given in Equation (10). Note that it uses the reference sending rate $s(t-D^R)$ without relying on a stored history of controls nor on exact Round-Trip delay measurements. All the information is rather extracted from the received feedback packets, and the media stream characteristics that are known at the sender. Furthermore, the information that is transmitted in the feedback packet can be accurately synthesized at each client even when media packets are lost, since all the packet headers contain the GOP size information $g_s(t)$. It is therefore sufficient to receive one media packet of the GOP for the client to generate an accurate feedback signal that stabilizes the control system.

Even if such a system is scalable and robust, it does not perform optimally yet due to network latency. Consider for example the scenario depicted in Figure 6(a). The feedback is only sent after a GOP n is completely received. Hence, it is available at the sender at the earliest one Round-Trip time after the last packet of the GOP has been transmitted. At the time the controller has to decide on the sending rate for the next GOP $n+1$, it does not have access to any feedback yet. It can thus only decide to keep on transmitting at the same sending rate. Even though an accurate feedback becomes available during the transmission of the GOP $n+1$, it can not be incorporated in the control system due to the structure of the video streams. Therefore, it can only affects the sending rate of the GOP $n+2$, which introduces a latency that is almost equal to the duration of one GOP.

In order to avoid such a latency that slows down the convergence of the control system to a steady-state solution, we propose to increase slightly the actual transmission rate. The data of a GOP with rate $s(t)$ is thus transmitted at a rate $s(t) + \frac{s(t) \cdot RTT}{1 - RTT}$. It therefore creates a guard-interval between the transmission of two adjacent GOPs n and $n+1$, so that the feedback information about the GOP n is likely to be received on time for controlling the sending of the GOP $n+1$. Figure 6(b) sketches the improved control sequence. The small increase in the transmission rate permits to increase the sending rate of the streaming session faster compared to the case illustrated in Figure 6(a).

VI. Simulation results

A. Setup

We illustrate now the behavior of the distributed rate allocation algorithm with simulation results in different streaming scenarios. We consider the general network topology depicted in Figure 7, where eight different sender-receiver pairs connect to a network through high-speed links but share a common bottleneck link. Each sender-receiver pair runs its own rate-control loop independently of the other pairs. It relies only on end-to-end feedbacks and the Utility information relative to

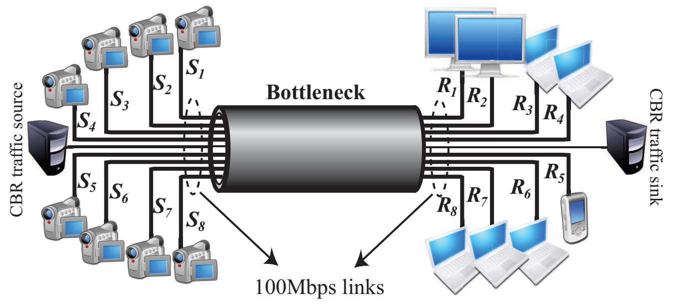

Fig. 7. Our simulation topology.

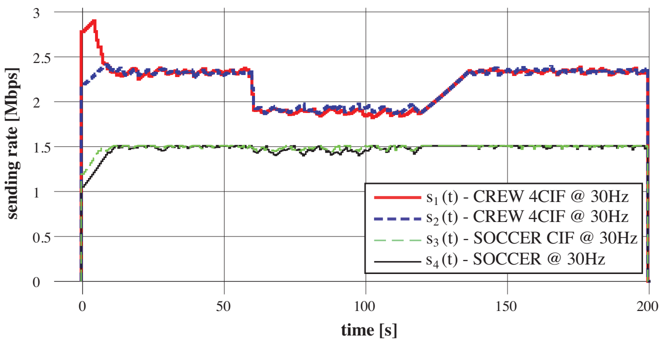

Fig. 8. Resulting sending rates/controls for Scenario 1.

the video sequence that is streamed. In particular, each sender and receiver knows neither the capacity of the bottleneck link, nor the number and nature of the other streams that are using the same bottleneck link. Finally, we also consider a constant bit rate (CBR) source-sink that also uses the same bottleneck link without any adaptive rate control.

The sequences transmitted by the senders are encoded using the H.264 SVC reference software (JSVM). In the encoding, we constrain each GOP of a given stream to span the same range of encoding rate. Unless otherwise stated, the Utility functions are given by the Rate-Distortion information extracted from the encoded sequences, as depicted in Figure 2. The distribution of the test video sequences between the different sender-receiver pairs is given in Table I for the scenarios considered in our simulations.

The proposed rate allocation algorithm is implemented in the application layer of the NS-2 simulator platform, and controls the rate of the underlying UDP transport protocol. The sender runs the control algorithm using the rate update equation (12) based on the utility functions of the transmitted stream, and the feedback it gathers from the receiver. Unless otherwise stated, the maximum loss rate factor π is set to 0.05 and the gain factor κ equals 1. The sender then extracts a SVC substream at the target sending rate with help of the tools available in the JSVM software distribution and sends the appropriate packets to the receiver. Upon reception of each packet, the receiver checks for losses and eventually reconstructs a received packet trace for each GOP. It forms a feedback packet based on he information it has received. The video sequence is finally decoded with the packets that have been correctly transmitted.

B. Adaptation to changing bottleneck capacity

In the first set of simulations, we analyze the effect of a changing bottleneck capacity, or equivalently the effect of varying background traffic in the Scenario 1. We set the bottleneck bandwidth to 8 Mbps and let senders S_1, S_2, S_3 and S_4 start transmitting simultaneously their respective video streams at time $t = 0$. After 60 seconds, we add a 1 Mbps CBR background traffic stream in the same bottleneck channel, which translates into a sudden bottleneck capacity reduction for the four sender-receiver pairs. The background traffic is switched off again after one further minute.

The resulting sending rates are shown in Figure 8 where they illustrate some key properties of the proposed algorithm. Clearly, the algorithm converges quickly to a stable state at the beginning of the transmission, as well as around the changes in background traffic. The stable state corresponds to maximization of the utilities of the different streams. Even though each stream is regulated independently of all the other ones, the same sequences converge to the same rate in the stable state. Once the CBR traffic joins the bottleneck, the streams react quickly to the new situation and settle in a new stable state almost immediately. Note that this behavior is very different from the sawtooth behavior seen in TCP for example. As the CREW sequences carried by S_1 and S_2 have a lower utility gradient than the SOCCER sequences transmitted by S_3 and S_4, the rates of the latter are hardly affected by the drop in bottleneck capacity. In other words, as the background traffic lowers the bottleneck bandwidth, the distributed control system allocates less rate to the CREW streams, as this results only in a minor overall quality degradation. A rate reduction

$S_{1,2} \rightarrow R_{1,2}$	CREW, 4CIF @ 30Hz
$S_{3,4} \rightarrow R_{3,4}$	SOCCER, CIF @ 30Hz
$S_5 \rightarrow R_5$	SOCCER, QCIF @ 15Hz
$S_{6,7} \rightarrow R_{6,7}$	ICE, CIF @ 30Hz
$S_8 \rightarrow R_8$	ICE, CIF @ 30Hz (Priority 1)

TABLE I
DISTRIBUTION OF THE TEST VIDEO SEQUENCES AMONG THE SENDER-RECEIVER PAIRS.

Scenario 1	Mean Y-PSNR at sender	Mean loss at receiver
$S_1 \rightarrow R_1$	33.30 dB	0.65 dB
$S_2 \rightarrow R_2$	33.28 dB	0.75 dB
$S_3 \rightarrow R_3$	36.62 dB	0.52 dB
$S_4 \rightarrow R_4$	36.56 dB	0.49 dB
Scenario 2		
$S_5 \rightarrow R_5$	36.12 dB	0.05 dB
$S_6 \rightarrow R_6$	41.06 dB	0.82 dB
$S_7 \rightarrow R_7$	41.24 dB	0.80 dB
Scenario 3		
$S_5 \rightarrow R_5$	34.01 dB	0.11 dB
$S_6 \rightarrow R_6$	39.99 dB	0.53 dB
$S_8 \rightarrow R_8$	41.92 dB	0.91 dB

TABLE II
PSNR QUALITY FOR THE DIFFERENT STREAMING SCENARIOS [dB].

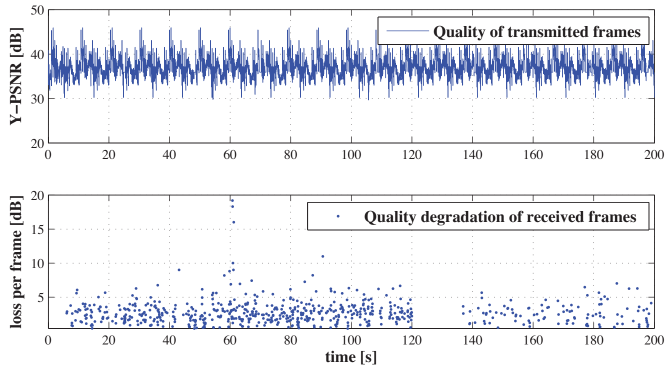

Fig. 9. Results for scenario 1. *Top:* Y-PSNR of each frame the SOCCER CIF@30Hz sequence transmitted by sender S_3. *Bottom:* Quality loss for each frame of the sequence, as received by client R_3.

for the SOCCER streams would result in a larger quality drop. Once the background traffic is switched off, the four streams return rapidly to their original stable rates. The average quality of the streams sent by the senders are reported in Table II in terms of Y-PSNR, along with the respective quality reduction due to packet loss during the transmission.

Finally, we show the temporal evolution of the quality for one of the test stream in Figure 9. We report the Y-PSNR quality for each frame of the stream transmitted by S_3. We also compute the quality loss at the receiver by substracting the Y-PSNR of each received frame from the Y-PSNR of the corresponding frame transmitted by the sender. Unsurprisingly, there is no loss when there is spare bandwidth available on the bottleneck link, i.e. during the first 10 seconds and after the CBR source switches off. There is however more packet loss due to congestion when the background traffic is active (i.e., in the timespan from 60 to 120 seconds). However, the steep gradient of the SOCCER Utility function prevents the rate from dropping. We note that loss cannot be completely avoided even in the steady state, as all the streams simultaneously compete for bandwidth shares until saturation of the bottleneck bandwidth. The small quality degradation resulting from these losses could be avoided by the use of error resiliency or error protection.

C. Adaptation to new streams

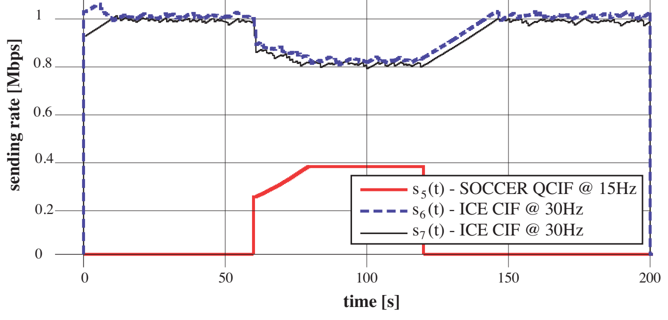

Fig. 10. Resulting sending rates/controls for Scenario 2.

In Scenario 2 we analyze the allocation of bandwidth resources when streams join and leave the bottleneck channel. We set the bottleneck bandwidth to 2 Mbps, and the sources S_6 and S_7 both start transmitting at time $t = 0$. After 60 seconds, the sender S_5 starts streaming during one minute, and then leaves the bottleneck again. The sending rates for this dynamic join/leave scenario are depicted in Figure 10.

It can again be seen that the system convergences rapidly into a stable state, where the two equivalent streams get an equal share of the bottleneck capacity. Once the the third source initiates its streaming sessions, the sending rates of the ICE sequences are gracefully brought down at S_6 and S_7 in order to adapt to the new situation. The new stream that contains the SOCCER QCIF sequence has a steep utility function and is therefore aggressive in getting shares of the bottleneck bandwidth. The quality of the transmitted streams and the corresponding quality drops due to packet loss are given in Table II.

In order to illustrate the benefit offered by a slight increase in the transmission rate, we have run the same simulation where the senders however do not implement the guard interval presented in the previous section (see Figure 6). The corresponding sending rates are given in Figure 11. It can be seen that in this case the algorithm has a slower convergence to the steady state due to the delay introduced by late feedbacks. This penalizes the average quality by about 1 dB per stream with respect to the same simulation scenario where the guard interval is used.

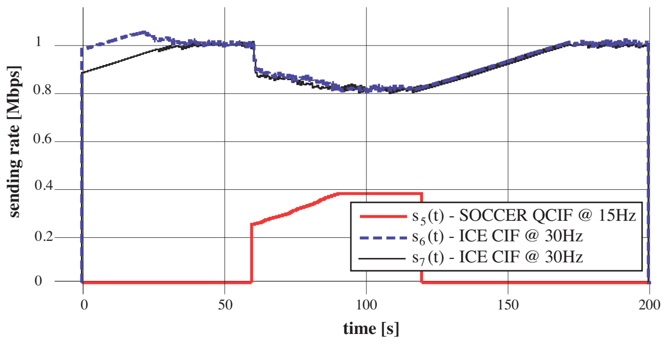

Fig. 11. Resulting sending rates/controls for Scenario 2 when the guard interval scheme is not used.

D. Adaptation to priority classes

As stated earlier, the concept of Utility function is very general and extends beyond the common characterization of video streams by their rate-distortion characteristics. To illustrate this, we consider in Scenario 3 a situation similar to the previous one, but where the 2 streams from S_6 and S_8 correspond to the same video content with different priority classes. They are characterized by the 2 Utility curves depicted in Figure 3, which could for example model two clients with differential treatment. The resulting sending rates for this simulation run are shown in Figure 12. They illustrate that the choice of Utility function directly drives the performance of the streaming application, which clearly favor the high priority

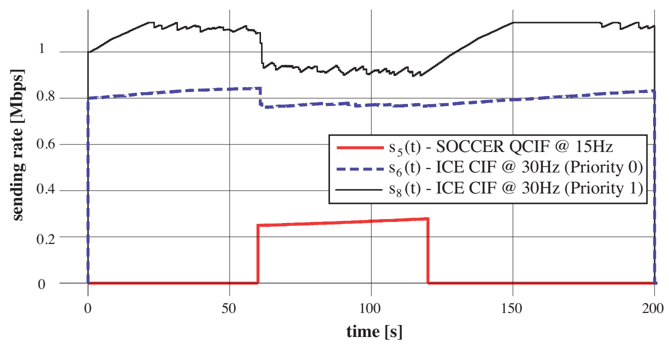

Fig. 12. Resulting sending rates/controls for Scenario 3. Both S_6 and S_8 transmit the same ICE sequence, but using different Utility curves.

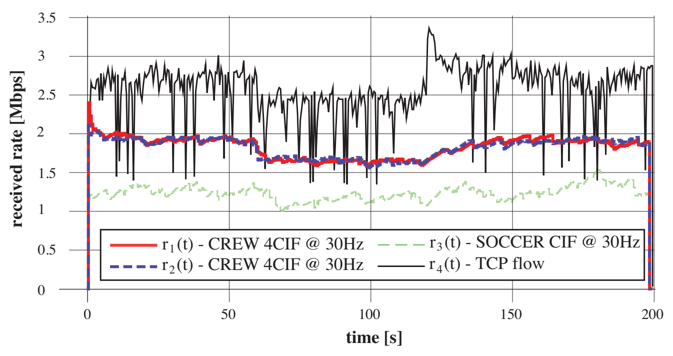

Fig. 14. Received rate in Scenario 1, where the stream from S_4 is replaced by a TCP-regulated flow.

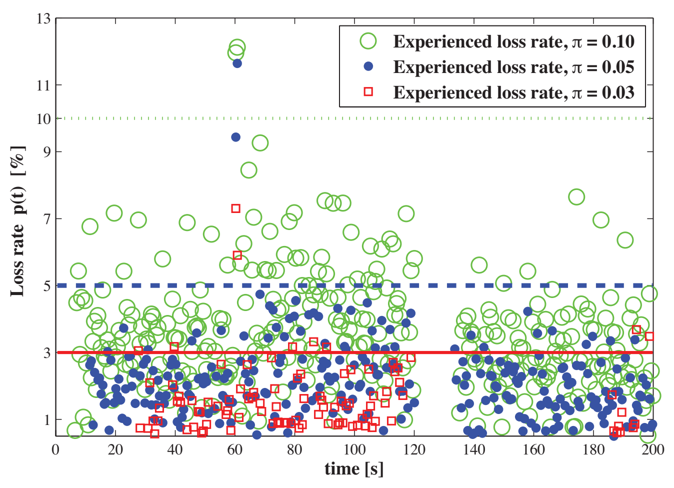

Fig. 13. Results for scenario 3: experienced loss rates $p(t)$, expressed in percentages, at receiver R_8 for different values of π

clients. We report again in Table II the average quality of the transmitted streams, along with the quality drops due to packet loss. We see that the high priority stream benefits from a 2dB quality gain compared to the low priority stream with the same video content.

Finally, we have run the same scenario several times using different values of π in the rate update Equation (12). Remember that this factor scales all the Utility gradients in the system and should thus bound the loss rate experienced by any stream in the stable state. The result of this simulation is given in Figure 13 where we show the experienced loss (which is equal to the congestion signal $p(t)$), as seen by receiver R_8 for the three cases where π equals 0.03, 0.05 and 0.1 respectively. As expected the packet loss are bounded by π in each of these cases. The parameter π is therefore an essential tool in order to adapt the rate allocation algorithm to a given decoder. A decoder can be characterized by a loss rate that it can cope with while decoding a stream, mostly through the use of error-concealment techniques. By matching π to the maximum tolerable loss rate at the decoder, we therefore ensure that the received stream is decodable when the system is in the stable state.

E. Adaptation to TCP traffic

We finally illustrate the behavior of the rate allocation algorithm when the bottleneck channel is shared with a flow controlled by TCP. We have considered a scenario similar to Scenario 1, but we replace the 4th stream in the system by an FTP data flow, which is regulated by TCP (Tahoe implementation). The results of this experiment are shown in Figure 14, which presents the received rates at each of the clients. The goodput of the TCP flow is shown as stream 4. Interestingly, these results show that our proposed congestion control protocol can coexist with TCP on the same bottleneck channel. Compared to the Scenario 1, the sending rates of the media streams converge slower and show slightly larger oscillations around their stable state values. This is due to the rapid changes in available bandwidth, which is mostly driven by the TCP flow. However, one can observe that the rates are still smooth and that the CREW and SOCCER streams are still handled appropriately, according to their respective Utility functions.

While this simulation does not represent any formal proof of proper distribution of resources between TCP flows, and the streams controlled by the algorithm proposed in this paper, it still shows that our algorithm does not starve TCP flows of the network resources. This corresponds to the results presented in [22], which state that the long term behavior of TCP Tahoe is equivalent to the one of a controller that maximizes a Utility function of the form $U(s) = \arctan(s)$. As this is a concave utility function, we should thus expect TCP Tahoe streams to be able to compete with streams regulated by any other concave Utility functions in a stable system.

Finally, we shall note that our algorithm is not *TCP-friendly* in the sense that the allocated rates yield an average per-flow bandwidth that is not equivalent to the one allocated by a TCP connection. TCP is in general more aggressive and tends to fairly share the average rate of each session, without considering the characteristics of each stream. Our algorithm targets a different objective, which is the effective allocation of the resources in order to maximize the average utility, or equivalently to make the best use of bandwidth resources in terms of application requirements.

VII. Conclusions and future work

We have presented a distributed rate allocation algorithm that targets the optimal distribution of bandwidth resources for improving the average quality of service of media streaming applications. We have first extended the framework of Network Utility Maximization with an important stability proof, which states that the algorithm is stable under general concave Utility functions and randomly delayed feedback. This result is particularly interesting for the implementation of rate allocation algorithms in real scenarios. We have then proposed an effective and scalable implementation of the distributed control algorithm with a light-weight application-layer protocol. We have further proposed a few practical utility functions for streaming video sequences. Finally, we have analyzed the behavior of the proposed solution with extensive NS-2 simulations, where we have considered H.264 SVC-FGS encoded sequences as an illustration. We have shown that the bandwidth allocation actually respects the constraints imposed by the utility functions, and that the system converges quite rapidly to a stable state after changes in the network. The proposed algorithm therefore provides an interesting solution for rate allocation in distributed streaming systems. We are currently investigating alternatives to further improve the convergence rate by relaxing the conservative rules used in the computation of the utility gradient. In addition, we are studying the extension of the framework to step-like Utility functions rather than concave functions, in order to provide efficient control solutions for a larger variety of streaming applications.

VIII. Acknowledgments

The authors would like to thank Dr Jacob Chakareski and Dr Jari Korhonen for fruitful discussions at the early stages of the work, and Dr Laurent Massoulié for feedbacks on the stability analysis for systems with heterogeneous delays.

References

[1] F. P. Kelly, “Charging and rate control for elastic traffic,” *European Transactions on Telecommunications*, vol. 8, pp. 33–27, 1997.

[2] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, “Rate control for communication networks: shadow prices, proportional fairness and stability,” *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252(16), Mar 1998.

[3] V. Jacobson, “Congestion avoidance protocol,” *ACM SIGCOMM*, pp. 314–329, 1988.

[4] Z. Cao and E. W. Zegura, “Utility max-min: an application-oriented bandwidth allocation scheme,” *IEEE INFOCOM*, vol. 2, pp. 793–901, Mar 1999.

[5] M. Chiang, S. Zhang, and P. Hande, “Distributed rate allocation for inelastic flows: Optimization frameworks, optimality conditions, and optimal algorithms,” *IEEE INFOCOM*, vol. 4, pp. 2679–2690, Mar 2005.

[6] J. Mo and J. Walrand, “Fair end-to-end window-based congestion control,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, 2000.

[7] W. H. Wang and S. H. Low, “Application-oriented flow control: Fundamentals, algorithms and fairness,” *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1282–1292, Dec 2006.

[8] Y. Zhang, S. R. Kang, and D. Loguinov, “Delayed stability and performance of distributed congestion control,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 307–318, Aug 2004.

[9] R. Johari and D. Tan, “End-to-end congestion control for the internet: Delays and stability,” *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 818–832, 2001.

[10] L. Massoulie and J. Roberts, “Bandwidth sharing: Objectives and algorithms,” *IEEE INFOCOM*, vol. 3, pp. 1395–1403, Jun 1999.

[11] S. Kunniyur and R. Srikant, “End-to-end congestion control schemes: Utility functions, random losses and ecn marks,” *IEEE INFOCOM*, vol. 3, pp. 1323–1332, 2000.

[12] P. Ranjan, R. J. La, and E. H. Abed, “Global stability conditions for rate control with arbitrary communication delays,” *IEEE/ACM Transactions on Networking*, vol. 14, no. 1, pp. 94–107, Feb 2006.

[13] R. J. La and V. Anantharam, “Utility-based rate control in the internet for elastic traffic,” *IEEE/ACM Transactions on Networking*, vol. 10, no. 2, pp. 272–286, Apr 2002.

[14] Y. Zhang and D. Loguinov, “Local and global stability of symmetric heterogeneously-delayed control systems,” *IEEE Conference on Decision and Control (CDC)*, 2004.

[15] T. Alpcan and T. Basar, “A globally stable adaptive congestion control scheme for internet-style networks with delay,” *IEEE/ACM Transactions on Networking*, vol. 13, no. 6, pp. 1261–1274, Dec 2005.

[16] J.-W. Lee, R. R. Mazumdar, and N. B. Shroff, “Non-convex optimization and rate control for multi-class services in the internet,” *IEEE/ACM Transactions on Networking*, vol. 13, no. 4, pp. 827–840, Aug 2005.

[17] L. Ying, E. Dullerud, and R. Srikant, “Global stability of internet congestion controllers with heterogeneous delays,” *American Control Conference*, vol. 4, pp. 2948–2953, Jun 2004.

[18] L. Massoulie, “Stability of distributed congestion control with heterogeneous feedback delays,” *IEEE Transactions on Automatic Control*, vol. 47, no. 6, pp. 895–902, Jun 2002.

[19] Y. Zhang, S. R. Kang, and D. Loguinov, “Delay-independent stability and performance of distributed congestion control,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, Dec 2007.

[20] M. Dai, D. Loguinov, and H. M. Radha, “Rate-distortion analysis and quality control in scalable internet streaming,” *IEEE Transactions on Multimedia*, vol. 8, no. 6, pp. 1135–1146, Dec 2006.

[21] J. Yan, K. Katrinis, M. May, and B. Plattner, “Media- and tcp-friendly congestion control for scalable video streams,” *IEEE Transactions on Multimedia*, vol. 8, no. 2, pp. 196–206, Apr 2006.

[22] J. He, M. Chiang, and J. Rexford, “Can congestion control and traffic engineering be at odds?” *Proceedings of the Global Telecommunications Conference. GLOBECOM ’06*, pp. 1–6, 2006.